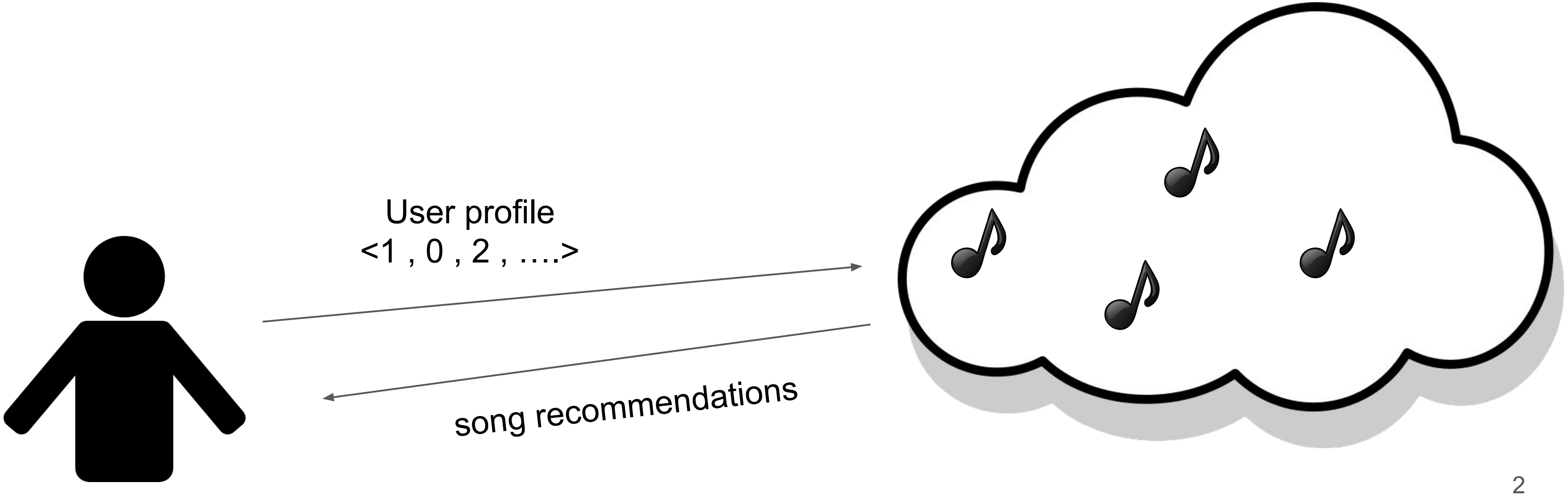
Privacy-Preserving Similarity Search Using Learned Indexes

MIT PRIMES Computer Science Conference
October 20, 2019

By: Patrick Zhang
Mentor: Kyle Hogan

Similarity Search

- matches *items* with similar *features* to the same user *profile*
  - each *item* has a *feature vector* - a vector of numbers determining certain qualities



Similarity Search

- Often used for online sites (e-commerce)
  - spotify
  - netflix
  - amazon

Examples of feature vectors for songs:

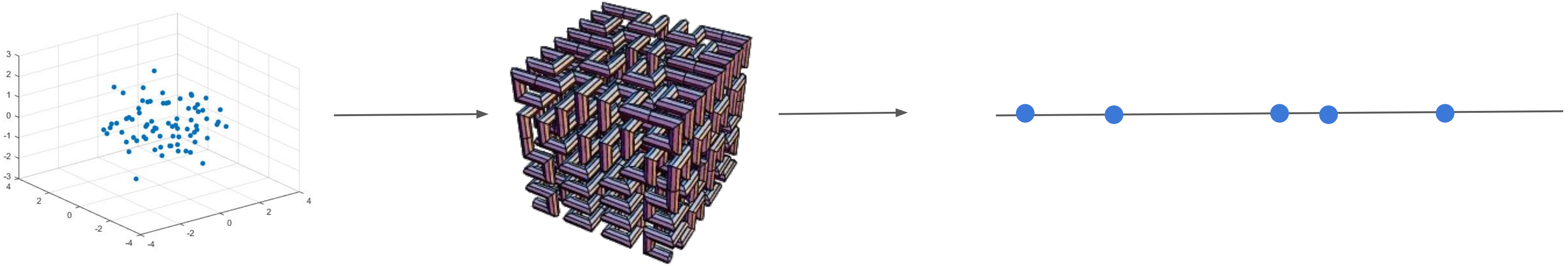
	Year	Is pop (genre)	Is jazz (genre)	length (seconds)
Song 1	2000	1 (yes)	0 (yes)	120
Song 2	2010	0 (no)	1 (yes)	200

Why make it private?

- Scenario
  - Client wants to get k song recommendations from Server, to match his profile
  - Both the Client and Server want privacy
    - Client doesn’t want the Server to know the profile (can contain very personal information)
    - Server doesn’t want the Client to learn the model that gives the song recommendations

Similarity Search Algorithm

- Each *feature vector* is a point on d-dimensional space where d is the size of a vector
    - Feature vector of song 1: <2000,1,0,120> has 4 dimensions
- k-nearest neighbors = k closest points to a single point
- Higher dimensions make it harder!
    - Map the points of d-dimensions to a 1 dimension using a Hilbert curve
    - Hilbert curve - a single space-filling line through d-dimensions that guarantees that if 2 points are close in 1 dimension, they will be close in d dimensions



Similarity Search Algorithm

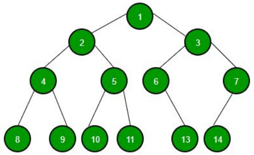
- All the items are on a single “sorted” array
  - <Song 1 (10 units) , Song 3 (100 units), Song 5 (101 units), ...>

- How can we find the index (location) of an item in the sorted vector privately?

What are Learned Index Structures

- Data structures to query information
  - we want to find the index of an item in an array

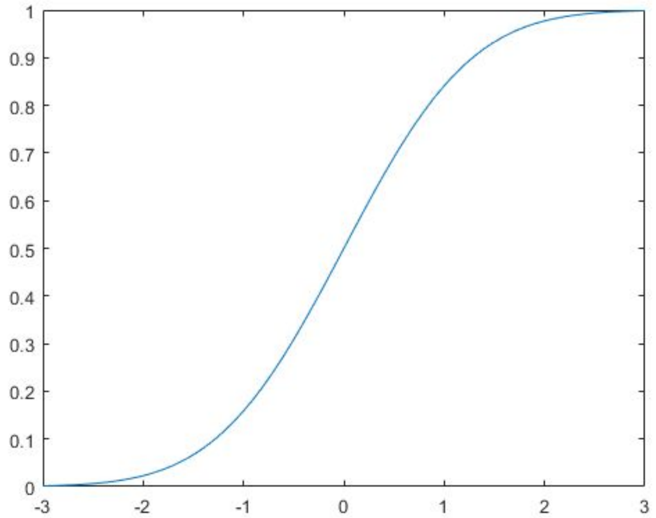
- How are these different from traditional index structures (i.e. Binary Trees)
  - they utilize the patterns in the data for an APPROXIMATE search that is more efficient in terms speed and memory



Creating a Learned Index Structure

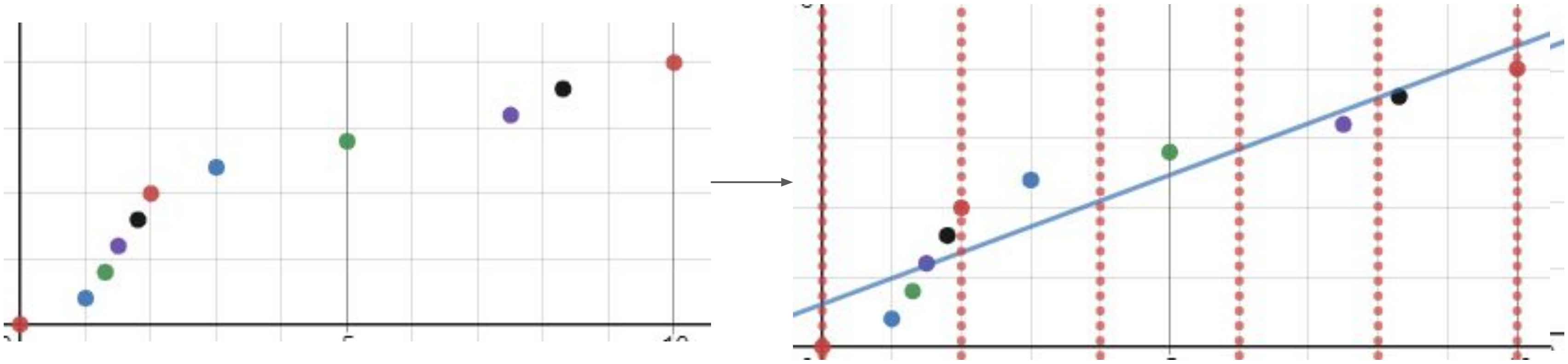
- want to approximate the position of a key in a sorted array
  - equivalent to approximating the CDF (cumulative distribution function (CDF)

  - x axis = distance
  - y axis = index



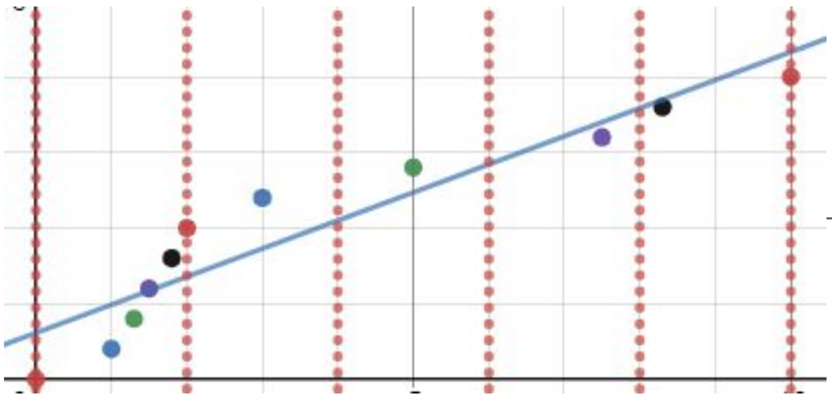
Creating a Learned Index Structure

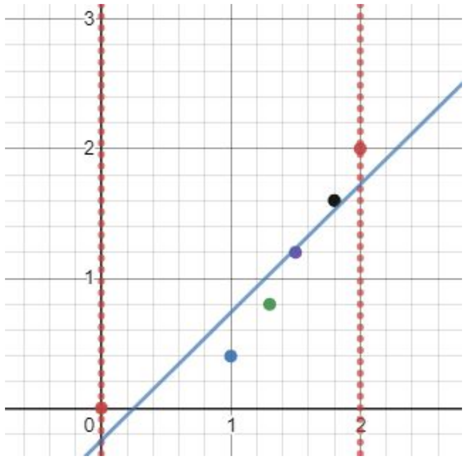
- use linear regression
  - find a line of best fit, x axis is the distance, y value is the position
  - however, there could be too much error
    - the result gives you a bin instead



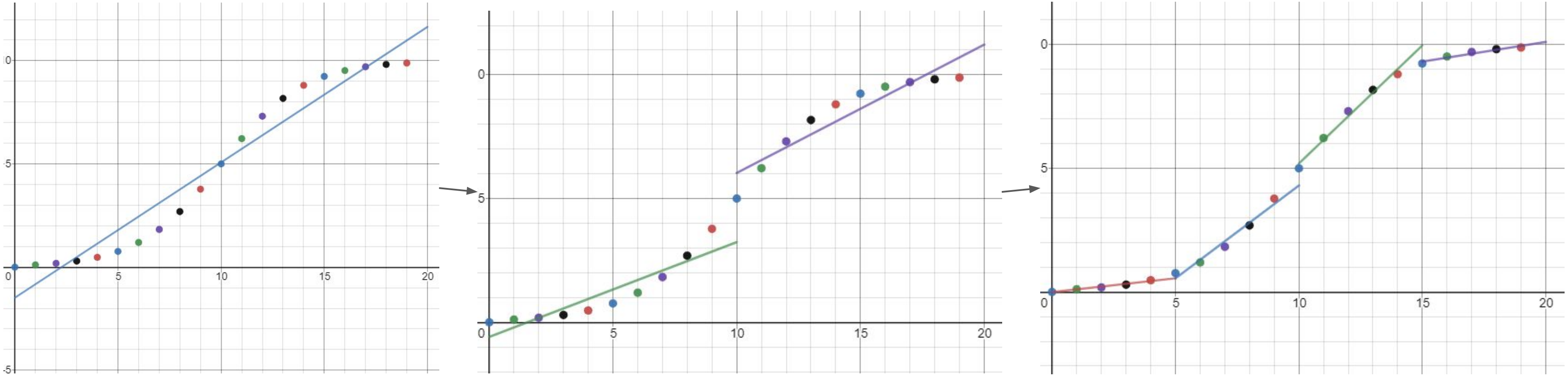
Creating a Learned Index Structure

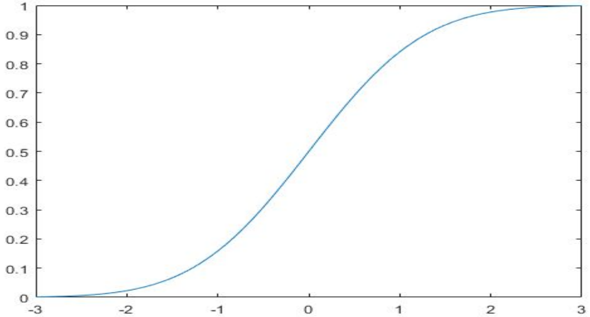
- within each bin, you find another line of best fit to find the approximate index
- each set of bins is a *layer,* each bin is represented by the equation of a line: y=mx+b





More layers = more bins = more accurate!



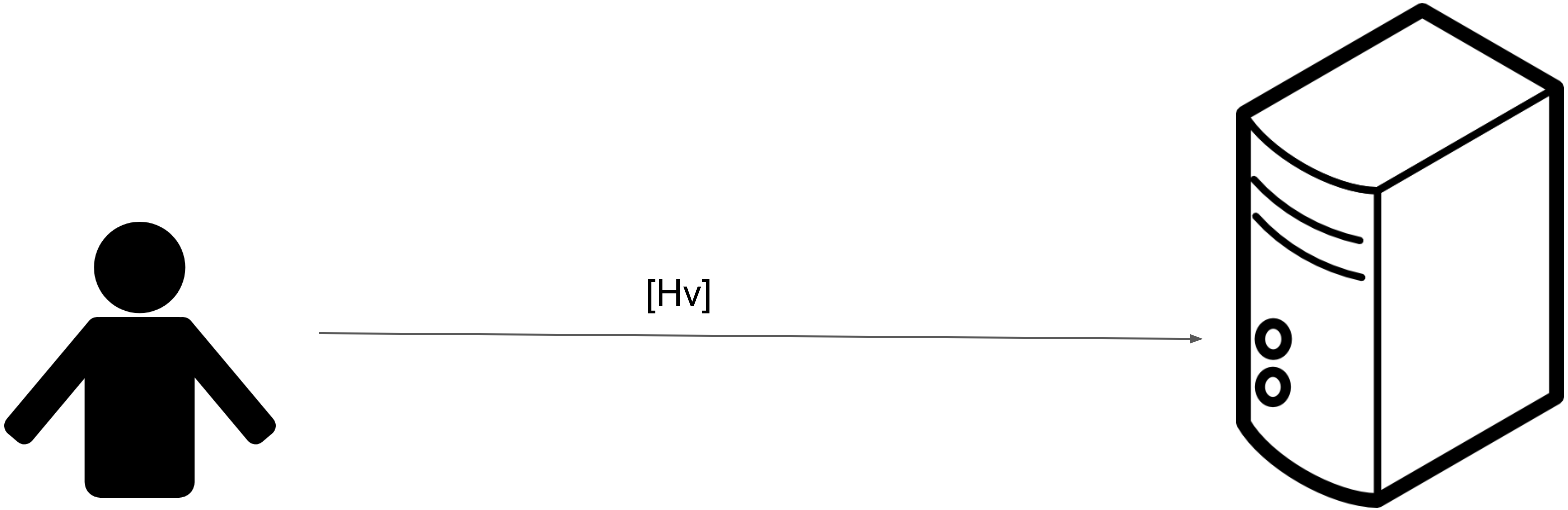


Current Protocol (Interactive) - Client Privacy

Client computes and encrypts Hilbert distance for profile [Hv] =

*[] means encrypted

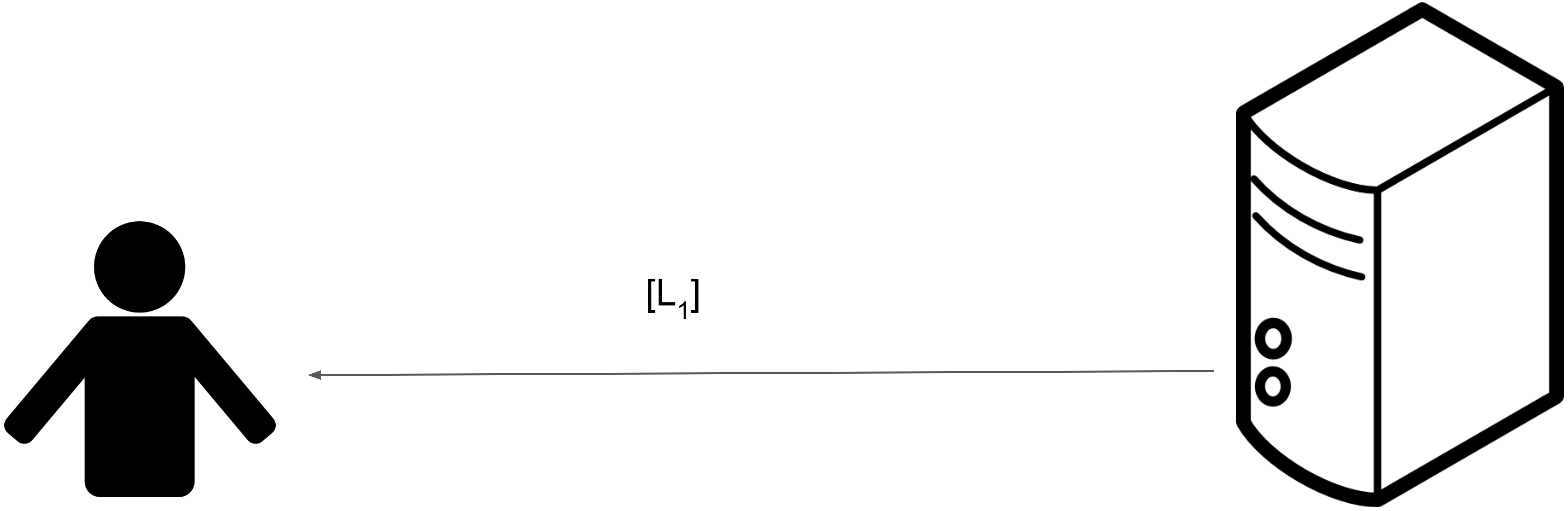
using somewhat homomorphic encryption (+ and * work under encryption i.e [x]*[y]=[xy])



Current Protocol (Interactive) - Client Privacy

Server uses line in layer 1 to get $[L_1]$, the encrypted result

$m[Hv]+b=[L_1]$

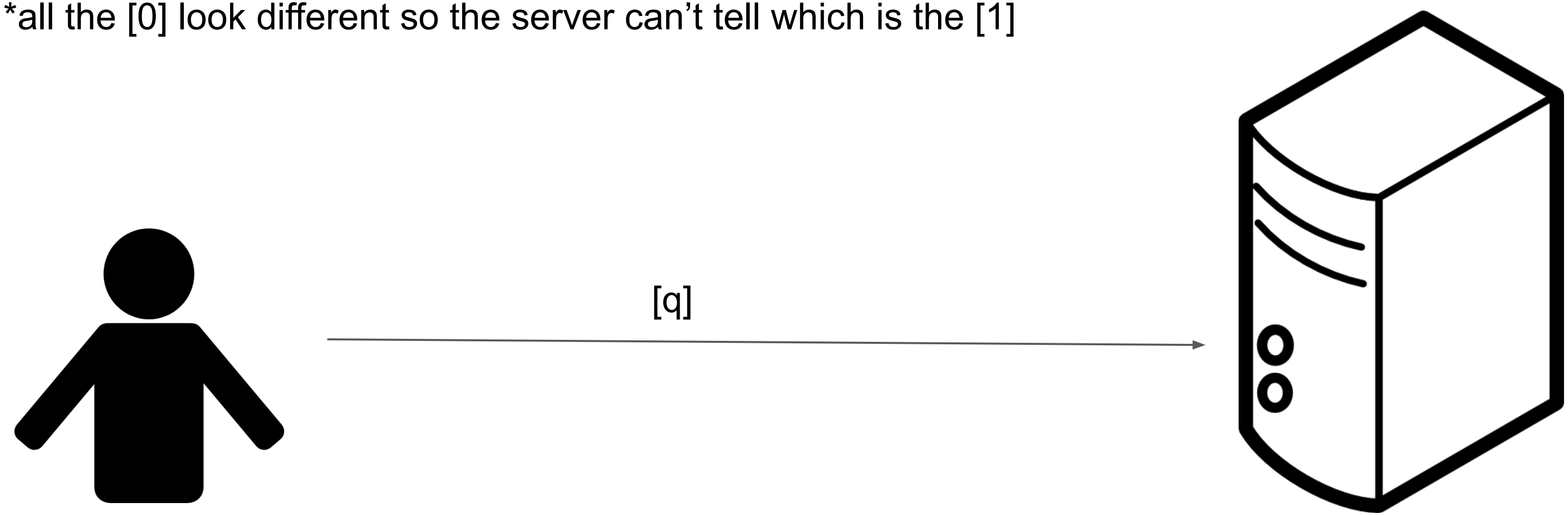


Current Protocol (Interactive) - Client Privacy

Client decrypts $[L_1]$ to get L_1, the index of the bin of layer 2

Finds vector $[q] = ([0],...,[1]....[0])$, array of $[0]$'s except for a $[1]$ at index L_1

*all the [0] look different so the server can't tell which is the [1]



Current Protocol (Interactive) - Client Privacy

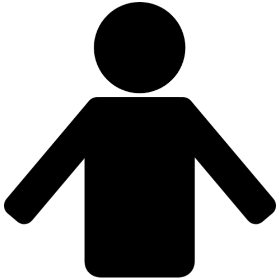
[q’] = ([1],...,[1])-[q] = ([1],....,[0],....,[1])

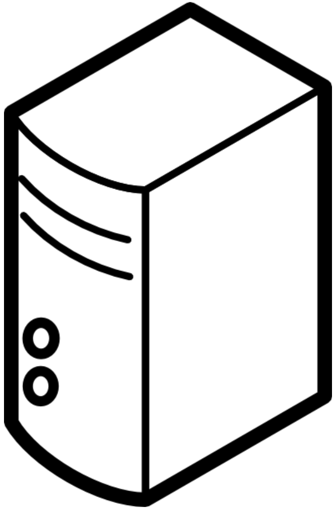
W = vector of x intercepts for next set of bins $(w_0,w_1,...)$
M = vector of slopes for next set of bins $(m_0,m_1,...)$
B = vector of y intercepts for next set of bins $(b_0,b_1,...)$

*lines for bins are represented as y=mx+b and mw+b=0

Server computes [s]=[q][Hv]+[q’]W
= $([w_0]$, $[w_1]$, ... , $[w_{i-1}]$, [Hv] , $[w_{i+1}]$,.....)

An array of the x intercepts except for [Hv] at L_1
(location of the bin we want)

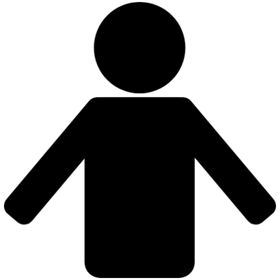


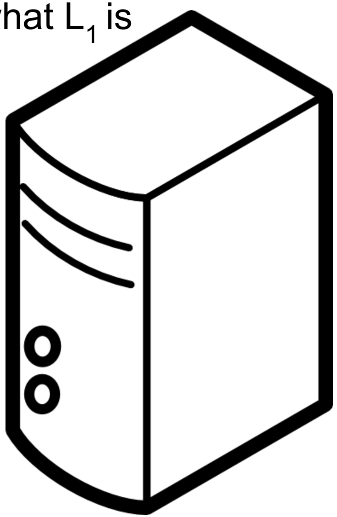


Current Protocol (Interactive) - Client Privacy

Server computes [s']=M[s]+B
$= (m_0[w_0]+b_0 , ... , m_i[Hv]+b_i, ...,m_n[w_n]+b_n)$
$= ([0], \dots , m_i[Hv]+b_i, \dots . [0])$

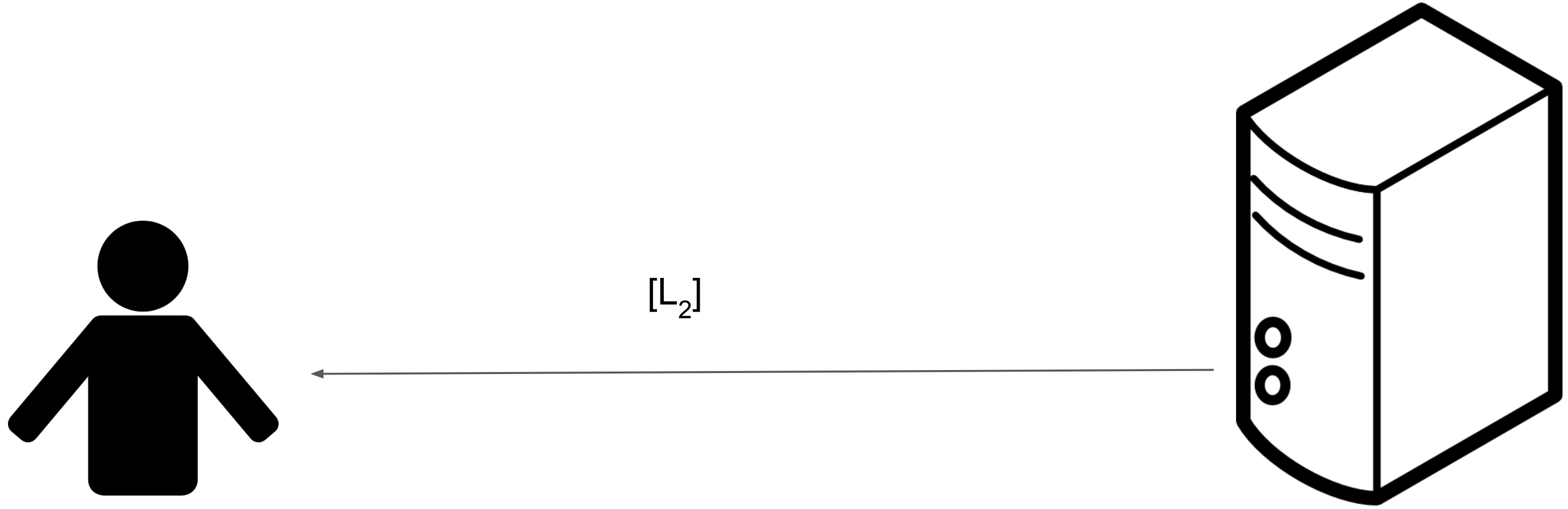
Remember mw+b=0
Still the server doesn't know what L_1 is





Current Protocol (Interactive) - Client Privacy

Server computes $[L_2]$ = sum of $[s']$ = $m_i[Hv]+b$



Current Protocol (Interactive) - Client Privacy

- Process is repeated until all layers in the model are processed (the last layer gives the final approximate index)
  - In the second layer, L_2 is used instead of L_1

Imminent Work (Adding Server Privacy)

- Server adds random number *r* to every L value (L_1,L_2..)
  - The Client doesn't know the actual index of the bins or the final index
- When the Server receives <...>, it rotates the values by *r*

Example:

L_1=1

r = 1

Client get L_1+ r =2 and sends: <[0], [0], [1]...>

Server rotates the values left by 1: <[0], [1], [0]...>

Future work

- Avoid making the Client compute the feature vector
  - the feature vector is also something that the Server often spends time making
  - we don’t know what features in songs spotify uses to determine similarity
- Decreasing bandwidth
  - the size of [q] can be big since it is equal to the number of bins in each layer, however in practice it's usually around 10 which is not so bad
- A problem to look into - only finds the index on a sorted array quickly, finding the k nearest neighbors requires PIR (Private Information Retrieval) - a really slow process for large databases (grows in speed proportionally to the size of the database)

Other Uses

- Even though similarity search may still be slow overall, privately querying indices of sorted arrays can be used for other things such as range queries

Acknowledgements

- MIT PRIMES

- My mentor: Kyle Hogan

- Assistance from: Sasha Servan-Schreiber and Hanshen Xiao

- My parents